Icy Rocks Workload: A Real Workload for the Android* Platform

Android Platform Benchmarks and Their Relevance to the User Experience

The Android OS, though having more than a billion users, is fairly new compared to Windows*, Linux*, and OSX*. As with any new system, how it works and how to improve it are often unclear. Further, users quickly and dramatically change their behavior and that trend shows no sign of slowing down.

In order to provide the best user experience, engineers create use cases, also known as workloads, that model how a real person uses the system, but are not themselves actual applications. Engineers can use such workloads to measure and improve the performance of Android system components.

Unfortunately, there are few such real-world workloads available for the Android platform. As a result, “unreal” and/or “trivial” synthetic workloads such as CaffeineMark* drive comparisons between devices and strongly influence what devices end up in the public’s hands. Such workloads tell us nothing about whether a real person would prefer one device over another, since they don’t measure anything about how devices are actually used. While optimizing such workloads may produce design wins, it does nothing to delight the end user.

Icy Rocks Workload Overview

Most Android Java* workloads attempt to measure one of two things. Some focus on a limited set of machine features and are designed to spotlight a specific component or set of components in the system on a chip (SoC). Others are not as biased toward specific machine features but consist of only a few Java methods. Both are synthetic and do not do the same things as code typically produced by application developers.

The Android execution stack is complex, so Android performance analysts should switch focus from such older workloads to new ones, which more accurately reflect Android end user activities. In order to provide end-user-perceivable improvements to the runtime, we require workloads that reflect the true characteristics of runtime and system-level interaction.

Icy Rocks Workload is an animation workload developed by Intel that mimics real-world game applications on the Android platform. It uses the open source JBox2D* physics engine and the Cocos2D* graphics engine, both written in Java. Cocos2D rendering is handled by OpenGL*. All object motions in Icy Rocks Workload are simulated by JBox2D. It simulates the physical game world and is responsible for world update.
Icy Rocks Workload incorporates several metrics. It collects the average number of Frames Per Second (FPS), commonly used to measure animation frame fluency, and average Animations Per Second (APS), which measures the calculation rate of the physics engine. Other metrics are average Frame Update Time and Jank. The final statistics are displayed on-screen and saved in a log file on the device.

**Game Simulation**

The primary Icy Rocks Workload interface is a scene in which there are many sprites such as rocks, snowflakes, a snowman, and a catapult. The catapult throws rocks at the snowman while rocks and snowflakes rain down into the valley between them. At the bottom of the valley is a rotating mixer stirring up the works. Game load is gradually increased by adding rocks and snowflakes.

Icy Rocks Workload runs in two different modes: demo and benchmark. In demo mode, the user can add rocks and snowflakes with each touch response. In benchmark mode, the workload runs automatically. Rocks and snowflakes are added every 20 seconds in a 2:5 ratio and screen touch is disabled. A measurement summary is available after five runs.

**Icy Rocks Workloads for the Android Platform, Java, and Native**

Icy Rocks Workload has three different variants for the Android platform, Oracle Java (JDK 8), and native C++. The workload animation has a similar appearance in all variations. There are two run modes: demo and benchmark. Demo mode runs continuously while benchmark mode runs for about 11 minutes. Icy Rocks Workload for the Android platform is the reference implementation. Icy Rocks Workload for Java can run on any Linux PC with Java 8 installed. Icy Rocks Workload for Native uses the C++ versions of Box2D and Cocos2D-X.
Both Icy Rocks Workload for the Android platform and Icy Rocks Workload for Java can be run in either the Graphics or the CPU (non-GL) modes. The CPU mode is also called Icy Rocks Workload for Kernel and is used to understand the standalone performance of the physics engine.

**How to Run the Icy Rocks Workload**

The Icy Rocks Workload for the Android platform is provided as one package: GameWorkload.apk. After installing GameWorkload.apk, the user clicks the icon for gameworkload launcher and selects either run benchmark or demo mode. The run benchmark is automated and the final scores are reported on screen upon completion. The demo mode supports user interaction.

**Game Benchmark Timeline**

In a typical real-world Android game, the complexity of the game increases as the user advances to higher levels and scores more points. In Icy Rocks Workload, we easily increase the load by adding more rocks and snowflakes as shown in the illustration below.

Among Android applications, FPS is the metric used to measure the smoothness of the user experience. Icy Rocks Workload measures the average number of frames it can render per second (FPS) at various load levels, then computes the final metric by taking the geometric mean of the FPS at the various load levels. The workload also measures frame drop rate, often referred as jank. Icy Rocks Workload measures jank per second (JPS) at various points in the animation.

A typical duration for Android mobile games is around 10 minutes, so the workload is designed to run for roughly 11. This time frame also accounts for a device ramp up time of 10 seconds between two different gaming configurations.

Each run reports the following metrics:

- Animations per second
- FPS
- Screen update time in seconds per frame
- Jank (in the Java world “Jank” means when the screen animations are not fluid and seem jumpy)

Once complete, an additional metric of the geometric mean of the data from these five runs is computed.

The run summary will look similar to this screenshot.

![Game Workload](image)

### Typical Game Workflow

![Game Workflow Diagram](image)

**Cocos2D for Android (Java) Workflow**
When using Cocos2D for the Android platform, the game is generally programmed via a Main Loop. The typical game workflow therefore resembles the figure above: it is defined by a main loop that cycles around: check if the user has interacted with the device, handle the game logic (moving objects, for example), prepare the animations and GL commands, and finally draw the frame.

As for all workloads, there is a requirement to provide metrics. For the Icy Rocks Workload, the various calculated metrics are FPS, APS, and JPS.

**JBox2d Workflow**

JBox2D is a 2D rigid-body simulation library. Programmers can use it to make objects move in realistic ways. JBox2D works as the game logic part in Cocos2D in the Icy Rocks Workload.

Games usually choose 1/60 seconds as world step interval. For the workload, JBox2D is used in Cocos2D to handle the physics part of the application.

It starts by defining all the objects in the world and setting up event listeners. Then, it initializes the world from a physics engine point of view and starts preparing its call-backs. At each step of the world, basically when the main loop calls it for an update, the JBox2D engine updates the positions and velocity of the objects it is tracking.

**Game Threads**

A generic game application contains two threads: a main and a renderer thread. The main thread initializes the application and game scene and then runs the main loop. The renderer thread is called at each drawing of the scene and also is in charge of calling any event listener. Here are the various details for both threads:

**Main thread**
- Initialize director (MainActivity.java)
- Initialize game scene (GameLayer.java)
  - Initialize physics objects in jbox2d world and sprites in cocos2d layer
  - Set contacts listener to handle special collisions

**Renderer thread**
- Default scheduler tick handler function (update() in GameLayer.java)
  - Step the world
  - Go through all physics objects and set the corresponding sprite position/angle
- **Contact listener** (postSolve in Gamelayer.java) for special effects such as:
  - Split a rock into grains when it hits heavy objects
  - Start score animation when a rock hits the snowman
- **Touch handler** (ccTouchesEnded in GameLayer.java) to add snowflakes/rocks
- **Metrics Drawing function** (showFPS in CCDirector.java)

The figure above shows in a graphical form how the drawing loop works. It resembles the main loop presented in the Cocos2D section but is a bit more complex. At each scheduler tick, the
new frame is calculated and updated. During that tick, callbacks and game logic are handled. For example, JBox2D is called to handle the game physics. Then, the rendering can begin.

Once the rendering is done, there can be a slight wait for the FrameTime. The wait is done in order to ensure consistent frame speeds. For example, if the application is able to calculate 100 FPS, it can actually slow itself down to render only 60 FPS. The rest of the time can be spent on more game logic, fetching data, and so on.

**Icy Rocks Workload Performance Overview**

Icy Rocks Workload is a single-threaded Android workload, referred to by the Android system as the “GLThread.” During a typical benchmark run most of time is spent in Android Runtime* (ART) compiled code and OpenGL native graphics routines. A smaller portion is spent in native code via the Java Native Interface (JNI) and System.arraycopy.

The workload showcases the capabilities of the Android Runtime, the graphics capabilities of the Android Stack and the hardware capabilities of the underlying SoC platform.

The Icy Rocks Workload for Java is expensive in terms of Floating Point Arithmetic and Logical (ALU) operations, Branch Prediction (floating point compares) hardware and DTLB and L2 cache memory transactions.

As mentioned earlier, the workload has five identical runs - during each run we spend 20 seconds in one particular configuration of Rocks + Snowflakes. Most of these configurations share identical performance characteristics (for example, instructions per cycle, hot java executed code, and so on). The workload is 32-bit single precision floating point intensive, which is the precision of the physics collision calculations.

**Opportunities Discovered Using Icy Rocks Workload (Open Source)**

During our performance investigation we discovered that the JBox2D physics engine uses its own implementation of the standard java.lang.Math library methods. In the current implementation huge data structures are maintained for Math.sin() and Math.cos() result lookups. We recommend changes in the JBox2D code to use the standard java.lang.Math library methods for optimal performance.

Our investigation also showed that JBox2D uses object pooling to avoid the runtime cost of object allocation and garbage collection. This may hurt GC performance in general, may cause heap fragmentation, and reduces object reference locality. Our recommendation is to use the garbage collector instead of object pooling. We further anticipate GC and reference locality improvements due to the use of object allocation based on Thread-Local Allocation Buffer (TLAB) in the relatively near future.
Additionally, Intel has contributed several optimizations to the Android Open Source Project (AOSP) and numerous ART optimizations have been added to Intel’s ART binaries as result of Intel’s performance analysis on Icy Rocks Workload. These will be described in a later article.

**Open Sourcing the Icy Rocks Workload for Android**

Open sourcing Icy Rocks Workload is part of Intel’s strategy to change the way Android performance is measured. The current set of synthetic benchmarks are, for the most part, unrealistic (they don’t reflect real-world app performance) and can sometimes be optimized away, as we recently did with CFBench* and Quadrant*. Our strategy is to persuade the Android community to drop synthetic benchmarks that can be optimized away and replace them with more realistic workloads such as Icy Rocks Workload.

**Download Locations**

Source code:  

APK released:  

**References**

cocos2d-android-1: [https://code.google.com/p/cocos2d-android-1/](https://code.google.com/p/cocos2d-android-1/)

What is Cocos2d-x?: [http://cocos2d-x.org/](http://cocos2d-x.org/)

Box2D: [http://Box2d.org](http://Box2d.org)

JBox2D: A Java Physics Engine: [http://Jbox2d.org](http://Jbox2d.org)

How to Make a Simple Android Game with Cocos2D:  

Intro to Box2D with Cocos2D 2.X Tutorial: Bouncing Balls:  

How to Make a Catapult Shooting Game with Cocos2D and Box2D Part 1:  

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at [intel.com](http://intel.com).

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request. Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting [www.intel.com/design/literature.htm](http://www.intel.com/design/literature.htm).

This sample source code is released under the [Intel Sample Source Code License Agreement](http://intel.com).

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© 2015 Intel Corporation.